Parallel and Distributed Systems - Kth Select Report

Spyridon Baltsas - AEM: 10443

1 Summary

This report is about my proposed solution for Kth Select problem. To be more specific, here are briefly presented and explained the algorithms I have implemented for my approach for both single process and multiple processes with distributed memory use cases. Additionally, the efficiency of MPI algorithm against various large datasets will be examined for different amount of processes, and MPI's effectiveness itself when it comes to distributing processes among several compute instances.

2 Algorithm

2.1 The problem

The problem is stated like this: "In a vector of unsorted elements, find which element would be in the k-th position if the vector was sorted in ascending order".

2.2 Approach

2.2.1 Sequential

The most effective algorithm to address this problem is QuickSelect. QuickSelect is very similar to QuickSort. A pivot element is selected from the array, and after using the `partition` function. The Lomuto's partitioning scheme was selected for the sake of simplicity. As a result, array is splitted in two sub-arrays;

- A sub-array containing elements which are smaller or equal to the pivot.
- A sub-array containing elements larger than pivot.

Here is the point where QuickSelect differs from QuickSort. Instead of recursing on both subarrays, we recurse on the subarray where the k-th is included. For example, if k is larger than the position of pivot, then we recurse to the sub-array containing the elements larger than pivot. Finally, the algorithm stops when k is equal to the position of pivot element. In that case, the problem is solved, we found the k-th smallest element.
It can be proven that the average time complexity is linear ($\mathcal{O}(n)$), and square ($\mathcal{O}(n^2)$) in the worst case where the array is already sorted.

2.2.2 Parallel / Distributed Memory

Since QuickSelect appears to be that fast when a single process is used, I attempted to create an implementation for distributed systems using the MPI library. In such a distributed memory environment, MPI is currently the only way to go. As a result, since shared memory between processes is completely absent, the task of implementing the quickselect algorithm in such a stable way may get very challenging for the following reasons;

- Starvation of root process. The process who controls the other processes (also known as the root process) by advertising the pivot, may run out of elements and therefore rely on the elements of the other processes to find the pivot element that should be picked correctly. As a result, there will be an overhead of processes communicating with each other, especially when processes are on different computers.
- Some processes may have all of the elements in their subarrays either larger or smaller as pivot, and because of that they may be not able to contribute to speeding up the search for the k-th element any further.
- Recursive nature of algorithm itself. There may be few extreme cases, sometimes very hard to predict, on which algorithm may fail to provide an answer, where either the answer will be incorrect or get stuck in an infinite loop. In addition, a recursive algorithm may not be able to get parallelized as efficiently as an imperative one could be.

DISCLAIMER: As the README of my repository states, the MPI version may get stuck in an infinite loop for a few cases. The reason is currently unknown. In case it happens, please try changing the number of processes, since it has been observed that fixes the issue most of the times.
Taking the above points into consideration the QuickSelect was modified. The behavior of algorithm changes slightly whether process is the controlling one or not. The logic can be summed up in the following steps;

1. Root process selects a pivot. If it has elements left, the last element of array is selected. Otherwise it is depleted, the smallest element of the rest of processes is selected as the pivot. The root process announces the pivot to the rest of processes.
2. All processes split their arrays again in 2 subarrays with the same partitioning scheme as the sequential, one containing elements less or equal to pivot (left sub-array), and one containing elements larger than pivot (right sub-array).
3. The amount of elements that is smaller than pivot and the amount of elements that are equal to that are calculated, in order the global position (the position of pivot in the sequential QuickSelect) of pivot to be found. Let p be the global pivot position and d be the amount of elements equal to pivot. If k belongs to the $[p, p + d)$ set, then the value of k-th element is found. The algorithm stops.
4. Otherwise, the following actions are taken;
   (a) If $k < p$;
      i. If process is the root and all elements are larger than pivot, process is depleted. Otherwise, find the position of the last element smaller than pivot and recurse into the resulting sub-array.
      ii. If the process is not the root, and the global position is out of the bounds of the current array, it is considered depleted and recurses into the same array (idles). Otherwise, if the element at the global position is equal to pivot, recurse on the left sub-array, excluding the element equal to pivot. If less than pivot, then again recurse on the left sub-array but contain the element as well.
   (b) If $k > p$, then similarly;
      i. If process is the root and all elements except pivot are less than it, process is depleted. Otherwise, find the position of the last element larger than pivot and recurse into the resulting sub-array.
      ii. If the process is not the root, and the global position is out of the bounds of the current array, it is considered depleted and recurses into the same array (idles). Otherwise, if the element at the global position is equal to pivot, recurse on the right sub-array, excluding the element equal to pivot. If larger than pivot, then again recurse on the right sub-array but contain the element as well.

3 Test Specifications

3.1 System Specifications

3.1.1 Sequential

3.1.2 Parallel / Distributed

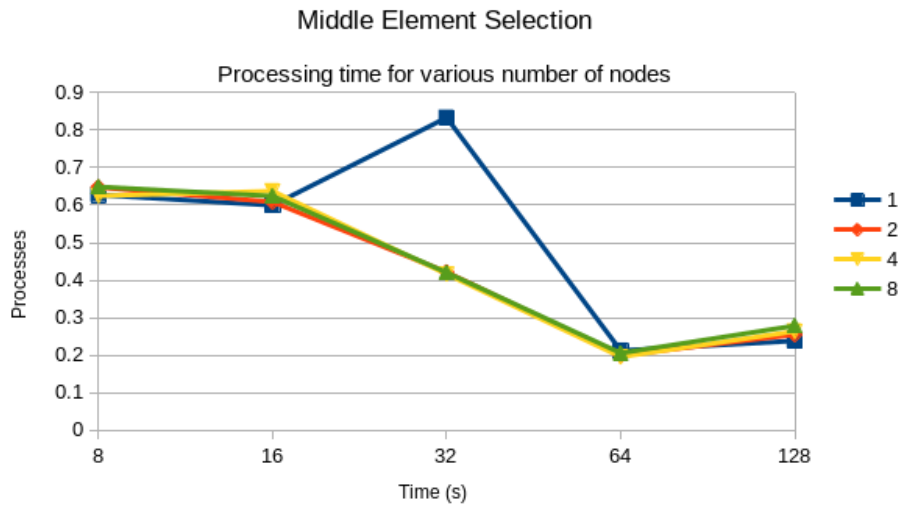
3.2 Test Dataset Specifications

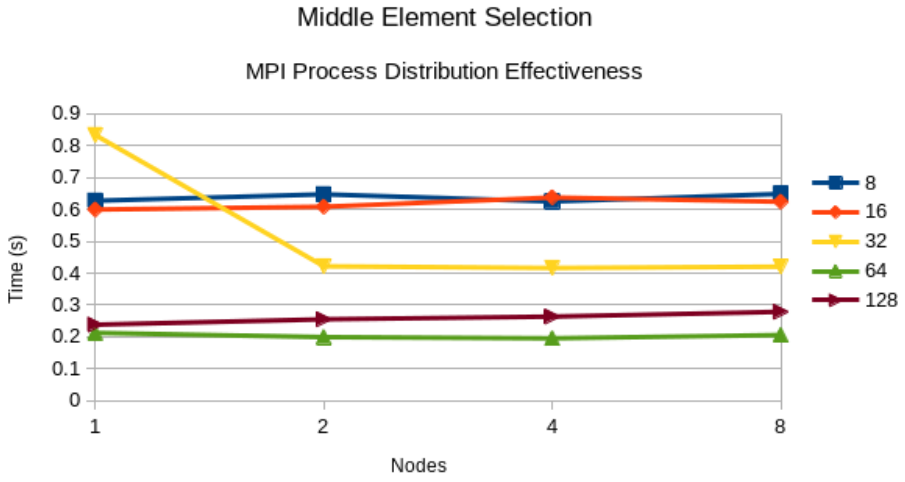
Name	File Size	32-bit Integer Count
Greek Wiki Dump	107 MB	28084359
NVIDIA CUDA 12 Linux Runfile	3.8 GB	1030846977

4 Results

4.1 Charts

4.1.1 Greek Wiki Dump





4.1.2 NVIDIA CUDA 12 Linux Runfile